

1 (20)	2 (50)	3 (30)	
20	45	30	95 (At)

Normas generales

- Numere las hojas entregadas. Complete en la primera hoja la cantidad total de hojas entregadas.
- Entregue esta hoja junto al examen, la misma **no** se incluye en la cantidad total de hojas entregadas.
- Está permitido tener los manuales y los apuntes con las listas de instrucciones en el examen. Está prohibido compartir manuales o apuntes entre alumnos durante el examen.
- Cada ejercicio debe realizarse en hojas separadas y numeradas. Debe identificarse cada hoja con nombre, apellido y LU.
- La devolución de los exámenes corregidos es personal. Los pedidos de revisión se realizarán por escrito, antes de retirar el examen corregido del aula.
- Los parciales tienen tres notas: I (Insuficiente): 0 a 59 pts, A- (Aprobado condicional): 60 a 64 pts y A (Aprobado): 65 a 100 pts. No se puede aprobar con A- ambos parciales. Los recuperatorios tienen dos notas: I: 0 a 64 pts y A: 65 a 100 pts.

Ej. 1. (20 puntos)

Responder detalladamente las siguientes preguntas, ejemplificar de ser posible.

- (4p) 1. Con segmentación *flat*, ¿Es posible utilizando solamente paginación, proteger una pagina de memoria para que no sea posible ejecutar código?
- (4p) 2. ¿Cuántos bytes de tamaño, tiene un segmento de límite 0 y granularidad 0?
- (4p) 3. ¿Qué diferencia hay entre el bit *dirty* y el bit *accessed* en una entrada de tabla de páginas?
- (4p) 4. ¿Cómo funciona un segmento *Expand Down*?
- (4p) 5. ¿Qué permisos efectivos tiene una pagina si su *Page Directory Entry* es de lectura/escritura con nivel de usuario y su *Page Table Entry* es de solo lectura con nivel supervisor?

Ej. 2. (50 puntos)

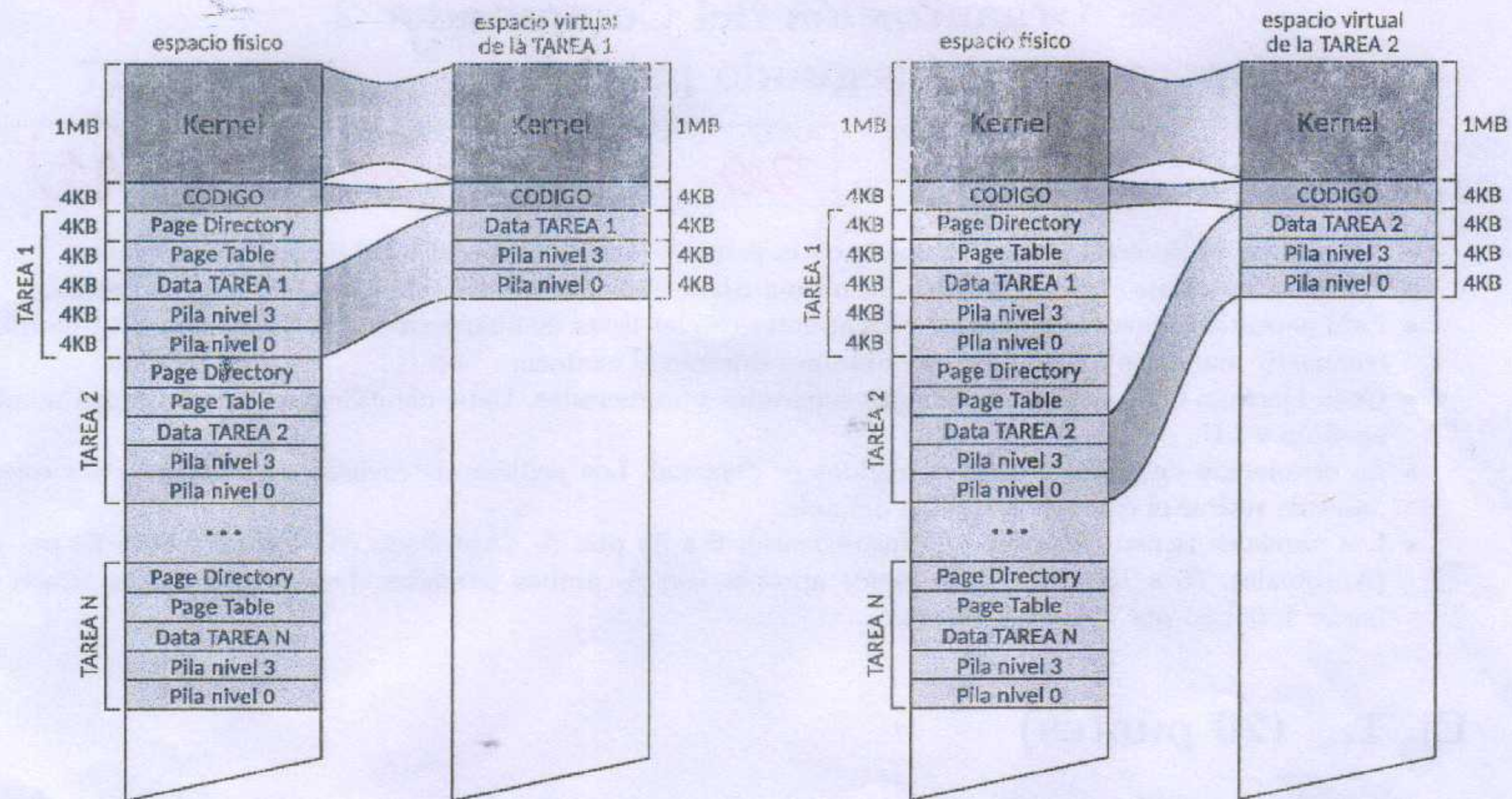
En un sistema tipo con segmentación *flat*, se propone el esquema de paginación que muestra la figura a continuación. Cada tarea ocupa exactamente 20 KB de memoria física, que corresponden a 5 paginas.

El código de las tareas será para todas el mismo, ocupado exactamente 4KB de memoria. El mapeo a memoria virtual de cada tarea corresponderá a mapear el código, datos y las dos pilas según corresponda a cada tarea.

Las tareas en el sistema son ejecutadas en orden, una por cada ciclo de reloj. Inicialmente las tareas comienzan con el EIP en la primer dirección de memoria de código y todos los registros en 0xFF, excepto el registro EAX, que contendrá el número de tarea que se esta ejecutando.

Las tareas pueden cometer cualquier tipo de excepción, en ese caso deben ser reiniciadas y comenzar a ejecutar inmediatamente como si ejecutadas por primera vez.

- (10p) 1. Indicar los campos relevantes de todas las estructuras involucradas en el sistema para administrar segmentación, paginación, tareas, interrupciones y privilegios. Instanciar las estructuras con datos y explicar su funcionamiento.
- (15p) 2. Programar en C la función *mapear_tarea*, que dado el puntero al directorio de paginas de una tarea se encarga de construir todo el mapa de paginación de la misma.
- (15p) 3. Programar en ASM/C la rutina de atención de interrupciones de alguna excepción del procesador.
- (10p) 4. Programar en ASM/C la rutina de atención de interrupciones del reloj.



Ej. 3. (30 puntos)

Suponer un sistema tipo con segmentación y paginación activa. Este sistema ejecuta tareas concurrentemente por cada ciclo de reloj. Las tareas pueden llamar a un servicio denominado `sendData`, que se encarga de copiar un buffer de memoria de una tarea cualquiera a otra diferente, incluso diferente de la tarea que llamo al servicio.

Los parametros de este servicio son los siguientes:

- `TASK_SRC`: nombre de la tarea fuente
- `TASK_DST`: nombre de la tarea destino
- `DIR_SRC`: dirección virtual en el espacio de la tarea fuente
- `DIR_DST`: dirección virtual en el espacio de la tarea destino
- `SIZE`: cantidad de bytes a copiar

Por cuestiones de performance, el servicio no puede modificar el registro `CR3`. Además se cuenta con la función: `uint32_t getCr3ByName(char* nombre)`, que toma el nombre de una tarea y retorna el `CR3` de la misma. El valor de `SIZE` no puede superar los 4000 bytes. Suponer que todas las tablas y directorios de página están mapeados con *identity mapping*.

- (15p) 1. Programar en C el código de la función `getFisica`, que dado un `CR3` y una dirección virtual válida, optiene la dirección de memoria física donde reside la misma.
- (15p) 2. Programar en ASM/C el código de la rutina de servicio `sendData`.

Nota: Considerar que todas las tareas tienen como espacio virtual libre desde la dirección `0x123000` a `0x234000`.

1)

1. Si, las páginas cuentan con un bit de protección contra ejecución (~~XD~~) (requiere PAE)

2. 1 byte, específicamente el byte al que apunta la dirección base. ✓

3. Una página con `accessed=1` indica que la misma fue leída, mientras que el bit `dirty` indica que fue escrita (en cache, debe hacerse escritura a memoria más adelante).

Esto permite manejar la memoria en caso de que haya múltiples CPUs. ✓

4. `expand-down` significa que los direcciones de memoria ~~convergen~~ del segmento son ~~formados~~ por el rango $(base - limit, base)$ en lugar de $(base, (base + limit))$. ✓

→

5. Por default, la página tiene permisos `rw - supervisor`.

El bit `CRO.WP` cambia esto a `read-only - supervisor`. ✓

2) 1. - Segmentación

Dado que se trata de segmentación flat, necesitamos 4 descriptores de segmentación:

GDT	índice	base	límite	DPL	TYPE	P	G	D/B	S
	1	0x0000	LIM	0	0x2	1	1	1	1
	2	"	"	0	0x4	1	1	1	1
	3	"	"	3	0x2	1	1	1	1
	4	"	"	3	0x4	1	1	1	1

$$LIM = 0x100 + 5 \times N \quad (\text{mínimo para } N \text{ tareas})$$

(256)

(máximo 0x3FF para poder usar 1 sola PT)
(esto es necesario para mapear, luego puede ser menor)

= Paginación

Dado que todas las tareas tienen en 1 sola PT con el kernel en identity mapping, se tiene 1 sola PDE (0x000) con los siguientes atributos:

URS = 1 (user)

RW = 1 (writable)

P = 1 (present)

base addr = (depende de cada tarea)

otros attrs = 0

La page table tiene los siguientes entradas:

- 0x000 - 0x0FF (kernel) ~~...~~

P = 1 (present)

base addr = índice de pt (identity mapping)

otros attrs = 0

- 0x100 (código tarea)

U/S = 1 (user)

P = 1 (present)

base addr = 0x100 (identifying mapping)

Otros bits = 0

0x101 - 0x102 (datos y pila nivel 3)

U/S = 1

R/W = 1

P = 1

base = (depende de la tarea)

Otros bits = 0

0x103 (pila lvl 0)

R/W = 1

P = 1

base-addr = (depende de la tarea)

Otros bits = 0

- manejo de tareas

Se necesitarán N descripciones de TSS en la GDT

más una TSS inicial para boot
~~(código que se ejecuta)~~

~~La TSS inicial~~. Las mismas tienen:

DPL = 0

P = 1

limit = 0x6f

Las mismas están ^{y en orden} configuradas en la GDT (2 fines prácticos)
se asume que el desc. de la TSS de la tarea 1 está en
el índice 8 de la GDT, y que la TSS básica está en el índice 7)

El kernel debe tener un arreglo de TSS (al que llamaré tareas) y una TSS básica con los siguientes datos:
(TSS_limpia)

- EDI, ESI, EBX, ECX, EDI = $0 \times FF$
- EIP = 0×100000
- EBP = $0 \times 102 \text{FFF}$
- ESP = $0 \times 102 \text{FFF}$
- EFLAGS = 0×202
- CS = 0×23
- DS, FS, FS, GS, SS = $0 \times 1B$
- SSD = 0×08
- ESP0 = $0 \times 103 \text{FFF}$
- IDMAP = $0 \times FF \text{FF}$

Los demás datos son rellenados al copiarla sobre otros TSS

- interrupciones

Necesitamos cargar una IDT que tenga ~~muchas~~ entradas para:

- todos los excepciones del procesador
- la interrupción del reloj (32)

Estos descripciones deben tener $p=1$ y $dpl=0$ para que no sean llamados por otros tareas

se asume que el PIC está configurado.

- Privilegios

El sistema propiamente solo utiliza 2 niveles de privilegios:
 0×0 (Kernel/supervisor) y 0×3 (User)

2.

```

void mapper_tarea (uint32_t cr3) {
    mapar_kernel(cr3);
    (pde*) pd = (pde*) cr3;
    mapar_kernel(pd);
    pt[0x100].us = 1;
    pt[0x100].p = 1;
    pt[0x100].base_addr = 0x100;

    pt[0x101].us = 1;
    pt[0x101].rw = 1;
    pt[0x101].p = 1;
    pt[0x101].base_addr = (cr3 + 0x2000) >> 12;

    pt[0x102].us = 1;
    pt[0x102].rw = 1;
    pt[0x102].p = 1;
    pt[0x102].base_addr = (cr3 + 0x3000) >> 12;

    pt[0x103].rw = 1;
    pt[0x103].p = 1;
    pt[0x103].base_addr = (cr3 + 0x4000) >> 12;
}

```

// se asume que esto se hace al principio, pero caso contrario
 // hay que limpiar el cache del TLB (Hobkirk)

```

void map2r-kernel (pte * p) {
    for (int i = 0; i < 0x100; ++i) {
        p + Σ i5 . p = 1
        p + Σ i5 . base_addr = i;
    }
}

```

3.

```

- isr 13:
    str eax
    push eax
    call restruccion_tarea
    pop eax

```

```

    mov [selector], eax

```

```

    ltr 0x38

```

```

    call km-intr-pic

```

```

    jmp f3 [offset]

```

; cambio a tss miod para recoger
; tss limpia

FALTA PORQUE EL BIT DE BUSY ESTA PRENDIDO

```

    dd offset 0

```

```

    dw selector 0

```



```
void reservar_tarea (uint32_t r) {
```

```
    uint32_t tarea = (r >> 3) - 1;
```

```
    tss * tss_tarea = & tarea2[tarea];
```

```
    memcpy((char *) tss_tarea, (char *) tss_tarea, 0x68);
```

```
    tss_tarea->eax = tarea;
```

```
    tss_tarea->cr3 = 0x10000 + 0x5000 * tarea;
```

```
}
```

también se puede leer el cr3
2020 d

```
void memcpy (char * src, char * dst, uint32_t len) {
```

```
    for (uint32_t i = 0; i < len; ++i) {
```

```
        dst[i] = src[i];
```

```
    }
```

```
}
```

4.

- 15r 32 i pushed.

~~pushad~~ ~~push eax~~

call sched_prox_tarea

~~add esp, 4~~

~~pop eax~~

~~le eax, 0~~

mov [Delegat], eax

call Km_intr_pic

jmp far [offset]

~~...~~

(selector y offset
de e. 3)

~~...~~

~~...~~

~~...~~

pop ad

iret


```
int_32t sched - prox - rarea (uint_32t tr - actual) {
```

```
int rarea_actual = (tr > 3) - 8;
```

```
return rarea_actual == N - 1;
```

P Ok 43 // loop, volver a rarea 1
: (tr - actual + 8); // continue con prox rarea

}

ESTO ES HORRIBLE.

LEER "THE C PROGRAMMING LANGUAGE"

DE K&R


```

3. Uint32_t + getFisica(Uint32_t cr3, Uint32_t virt) {
    pde *pd = (pde *) cr3;

    Uint32_t pd_off = virt >> 22;
    Uint32_t pt_base = pd[pd_off].base_addr;
    pte *pt = (pte *) pt_base << 12;
    Uint32_t pt_off = (virt << 10) >> 22;
    Uint32_t phys_base = pt[pt_off].base_addr;
    return (phys_base << 12) + (virt & 0xFFF);
}

```

2. Se asume que el servicio es un interrupt gate que tiene los siguientes parámetros en los siguientes registros:

```

eax: task_ptr
ebx: task_ptr
ecx: dir_ptr
edx: dir_ptr
esi: size

```



```

send_data:
    pushad
    push edi, cr3
    push esi
    push edi
    push ecx
    push ebx
    push eax
    call copiar_datos
    add esp, 24
    popad
    ret

```

```

void copiar_datos(char * task_src, char *task_dst, uint32_tdir_src,
                 uint32_tdir_dst, uint32_tsize) {
    uint32_t cr3_src = getCr3ByName(task_src);
    uint32_t cr3_dst = getCr3ByName(task_dst);

    uint32_t fis_src = getFisic(cr3_src, dir_src);
    uint32_t fis_dst = getFisic(cr3_dst, dir_dst);

    mapear_pagina(fis_src, 0x123000, cr3);
    mapear_pagina(fis_dst, 0x124000, cr3);
    mapear_pagina(fis_src, 0x125000, cr3);
    mapear_pagina(fis_dst, 0x126000, cr3);
    mapear_pagina(fis_src, 0x127000, cr3);

    desmapear_pagina(0x123000, cr3);
    desmapear_pagina(0x124000, cr3);
    desmapear_pagina(0x125000, cr3);
    desmapear_pagina(0x126000, cr3);
}

```

NO HACE FALTA

```

* {
    uint32_t virt_src = 0x123000 | (cr3_src & 0xFFF);
    uint32_t virt_dst = 0x125000 | (cr3_dst & 0xFFF);
    memcpy(virt_src, virt_dst, cr3);
}

```

El trabajo consistía de que el buffer ocupase más de 1 página

NOTA (max 2 por el máximo size)


```

void mapear_pagina (uint32_t fis, uint32_t virt, uint32_t cr3) {
    pde * pd = (pde *) cr3;
    uint32_t pd_off = virt >> 22;
    pd[pd_off].p = 1;
    pd[pd_off].rw = 1;
    uint32_t
    uint32_t pt_base = pd[pd_off].base_addr;
    pte * pt = (pte *) pt_base << 12;
    uint32_t pt_off = (virt << 10) >> 22;
    pt[pt_off].p = 1;
    pt[pt_off].rw = 1;
    pt[pt_off].base_addr = fis >> 12;
    flush_tlb flush_tlb();
}

```

```

void desmapear_pagina (uint32_t virt, uint32_t cr3) {
    pte * pt = (pte *) cr3;
    uint32_t pt_off = virt >> 22;
    uint32_t
    uint32_t pt_base = pt[pt_off].base_addr;
    pte * pt = (pte *) pt_base << 12;
    uint32_t pt_off = (virt << 10) >> 22;
    pt[pt_off].p = 0;
}

```



```
void memcpy ( unsigned char * src, unsigned char * dst, unsigned int len) {
```

```
    char * ssrc = (char *) src;
```

```
    char * ddst = (char *) dst;
```

```
    for (int i = 0; i < len; ++i) {
```

```
        ddst[i] = ssrc[i];
```

```
    }
```

```
}
```

